**In the Claims:**

Please amend claims 1, 2, 10, 11, 19-25, 32 and 33 as shown below.

1. (Currently amended) A resource access control mechanism for a multi-thread computing environment, the mechanism being operable:

to manage a sequence of one or more mutexes, wherein the sequence of mutexes is associated with a resource; and

when a requesting thread attempts an access to the resource[[,]]:

to lock a mutex, wherein the locked mutex is allocated to the requesting thread[[,]];

to make a determination whether the sequence includes a previous mutex; and

if a result of the determination is positive, to attempt to lock [[a]] the previous mutex in the sequence ~~if present~~, ~~whereby~~ wherein the requesting thread is suspended if the previous mutex is already locked until the previous mutex is unlocked in response to a previous thread finishing access to the resource.

2. (Currently amended) The mechanism of claim 1, the mechanism being operable, on attempting to lock [[a]] the previous mutex in the sequence when the previous mutex is unlocked, to lock the previous mutex on behalf of the requesting thread and then to unlock the previous mutex on behalf of the requesting thread.

3. (Original) The mechanism of claim 1, wherein the resource access control mechanism unlocks the mutex allocated to the requesting thread in response to the requesting thread completing access to the resource.

4. (Original) The mechanism of claim 1, wherein the mechanism includes an internal mutex operable to protect the locking of the mutex allocated to the requesting thread.

5. (Original) The mechanism of claim 1, wherein the resource comprises a print function.

6. (Original) The mechanism of claim 1, wherein the sequence of mutexes is held in an array.

7. (Original) The mechanism of claim 1, wherein the sequence of mutexes is held in a ring buffer.

8. (Original) The mechanism of claim 1, wherein the sequence of mutexes is held in a linked list.

9. (Original) The mechanism of claim 1, wherein the sequence of mutexes is held in a circular linked list.

10. (Currently amended) A resource access control program for a multi-thread computing environment, the program comprising program code on a carrier medium, which program code is operable to manage a sequence of mutexes, wherein the sequence of mutexes is associated with the resource and is operable to respond to a call from a thread requesting access to a resource by:

locking a mutex for the requesting thread, wherein the mutex is allocated to the requesting thread;

making a determination whether the sequence includes a previous mutex; and

if a result of the determination is positive, attempting to lock [[a]] the previous mutex in the sequence ~~if present~~, ~~whereby~~ wherein the requesting thread is suspended if the previous mutex is already locked, until the previous mutex is unlocked in response to a previous thread finishing access to the resource.

11. (Currently amended) The program of claim 10, wherein the program code is operable, on attempting to lock [[a]] the previous mutex in the sequence when the previous mutex is unlocked, to lock the previous mutex on behalf of the requesting thread and then to unlock the previous mutex on behalf of the requesting thread.

12. (Original) The program of claim 10, comprising program code operable to respond to a call from the requesting thread completing access to the resource by unlocking the mutex allocated to the requesting thread.

13. (Original) The program of claim 10, wherein the program code is operable to control an internal mutex operable to protect the locking of the mutex allocated to the requesting thread.

14. (Original) The program of claim 10, wherein the resource comprises a print function.

15. (Original) The program of claim 10, wherein the sequence of mutexes is held in an array.

16. (Original) The program of claim 10, wherein the sequence of mutexes is held in a ring buffer.

17. (Original) The program of claim 10, wherein the sequence of mutexes is held in a linked list.

18. (Original) The program of claim 10, wherein the sequence of mutexes is held in a circular linked list.

19. (Currently amended) A computer program product comprising program code on a ~~carrier~~ tangible computer readable medium, which program code is operable to manage a sequence of one or more mutexes, wherein the sequence of mutexes is associated with a resource, and the program code is further operable to respond to a call from a thread requesting access to the resource by:

locking a mutex for the requesting thread, wherein the mutex is allocated to the requesting thread;

making a determination whether the sequence includes a previous mutex; and

if a result of the determination is positive, attempting to lock [[a]] the previous mutex in the sequence ~~if present~~, ~~whereby~~ wherein if the previous mutex is already locked, the requesting thread is suspended until the previous mutex is unlocked in response to a previous thread finishing access to the resource.

20. (Currently amended) The computer program product of claim 19, wherein the ~~carrier~~ tangible computer readable medium comprises a storage medium.

21. (Currently amended) The computer program product of claim 19, wherein the ~~carrier~~ tangible computer readable medium comprises a transmission medium.

22. (Currently amended) A computer system, comprising:

a processor;

a memory storing program instructions for a method for controlling access to a resource for a multi-thread computing environment, wherein upon execution of said ~~method~~ program instructions on said processor said method comprises:

managing a sequence of one or more mutexes, wherein the sequence of mutexes is associated with the resource;

receiving a request from a thread to access the resource;

locking a mutex in the sequence for the requesting thread, wherein the mutex is allocated to the requesting thread;

making a determination whether the sequence includes a previous mutex; and

if a result of the determination is positive, attempting to lock [[a]] the previous mutex in the sequence ~~if present~~, ~~whereby~~ wherein the requesting thread is suspended if the previous mutex is already locked until the previous mutex is unlocked in response to a previous thread finishing access to the resource.

23. (Currently amended) The computer system of claim 22, wherein upon execution of said program instructions on said processor, said method further ~~comprising~~ comprises, after attempting to lock [[a]] the previous mutex in the sequence:

locking the previous mutex on behalf of the requesting thread when the previous mutex is unlocked; and

unlocking the previous mutex on behalf of the requesting thread ~~when~~ after the previous mutex is locked on behalf of the requesting thread.

24. (Currently amended) The computer system of claim 22, wherein upon execution of said program instructions on said processor, said method further ~~comprising~~ comprises: unlocking the mutex allocated to the requesting thread in response to the requesting thread completing access to the resource.

25. (Currently amended) The computer system of claim 22, wherein upon execution of said program instructions on said processor, said method further ~~comprising~~ comprises: locking an internal mutex operable to protect the locking of the mutex allocated to the requesting thread.

26. (Original) The computer system of claim 22, wherein the resource comprises a print function.

27. (Original) The computer system of claim 22, wherein the sequence of mutexes is held in an array.

28. (Original) The computer system of claim 22, wherein the sequence of mutexes is held in a ring buffer.

29. (Original) The computer system of claim 22, wherein the sequence of mutexes is held in a linked list.

30. (Original) The computer system of claim 22, wherein the sequence of mutexes is held in a circular linked list.

31. (Original) The computer system of claim 22, wherein the method stored in the memory comprises a computer program.

32. (Currently amended) A method of resource access control for a multi-thread computing environment, the method comprising:

managing a sequence of one or more mutexes, wherein the sequence of mutexes is associated with a resource;

receiving a request from a thread to access the resource;

locking a mutex in the sequence for the requesting thread[[,]];

making a determination whether the sequence includes a previous mutex; and

if a result of the determination is positive, attempting to lock [[a]] the previous mutex in the sequence ~~if present~~, ~~whereby~~ wherein the requesting thread is suspended if the previous mutex is already locked until the previous mutex is unlocked in response to a previous thread finishing access to the resource.

33. (Currently amended) The method of claim 32, further comprising, on attempting to lock [[a]] the previous mutex in the sequence when the previous mutex is unlocked, locking the previous mutex on behalf of the requesting thread and then unlocking the previous mutex on behalf of the requesting thread.

34. (Original) The method of claim 32, further comprising unlocking the mutex allocated to the requesting thread in response to the requesting thread completing access to the resource.

35. (Original) The method of claim 32, further comprising managing an internal mutex to protect the locking of the mutex allocated to the requesting thread.

36. (Original) The method of claim 32, wherein the resource comprises a print function.

37. (Original) The method of claim 32, wherein the sequence of mutexes is held in an array.

38. (Original) The method of claim 32, wherein the sequences of mutexes is held in a ring buffer.

39. (Original) The method of claim 32, wherein the sequence of mutexes is held in a linked list.

40. (Original) The method of claim 32, wherein the sequence of mutexes is held in a circular linked list.